实验六 综合案例（4）

一、 实验目的

1、 掌握继承和派生的定义与操作

二、 实验内容

1、PTA 作业《C++语言程序设计工程实践：面向过程高级》实验题

2、使用 Visual Studio 环境来编辑、编译和运行下列程序。

设计基类 Account 用来描述所有账户的共性，然后将 SavingsAccount 类变为其派生类,此外再从中派生出用来表示信用账户的类 CreditAccount。

在基类 Account 中,需要保留的数据成员是表示账号的 id、表示余额的 balance 和表示账户总金额的静态数据成员 total,以及用于读取它们的成员函数和用来输出账户信息的 show 函数。原有的 record 和 error 成员函数的访问控制权限修改为 protected,因为派生类需要访问它们,此外,需要为它设置一个保护的构造函数,供派生类调用。

用来处理存款的成员函数 deposit,用来处理取款的成员函数 withdraw,用来处理结算的成员函数 settle 都被放到了各个派生类中,原因是两类不同账户存取款的具体处理方式不尽相同。此外,储蓄账户用来表示年利率的 rate、信用账户用来表示信用额度的 credit,表示日利率的 rate、表示年费的 fee 以及用来获取它们的成员函数都作为相应派生类的成员。此外,虽然 Account 中存在用来输出账户信息的 show 函数,但对于信用账户而言,我们希望在输出账户信息的时候,除输出账号和余额外,将可用信用额度一并输出，因此为 CreditAccount 类定义了同名的 show 函数，而该类的 getAvailableCredit 函数就用来计算可用的信用额度。

两类账户的利息计算具有很大的差异,无论是计息的对象还是计息的周期都存在差异,因此计息的任务也不能由基类 Account 完成。然而,两类账户在计算利息时都需要将某个数值(余额或欠款金额)按日累加,为了避免编写重复的代码,有如下两种可行的解决方案。

(1)在基类 Account 中设立几个保护的成员函数来协助利息计算，类似于第 6 章的 SavingsAccount 类中的成员函数 accumulate,此外还需包括修改和清除当前的按日累加值的函数，然后在派生类中通过调用这些函数来计算利息。

(2) 建立一个新类,由该类提供计算一项数值的按日累加之和所需的接口,在两个派生类中分别将该类实例化,通过该类的实例来计算利息。

由于计算一项数值的按日累加之和这项功能是与其他账户管理功能相对独立的,因此将这项功能从账户类中分离出来更好,即采用第(2)种解决方案。这样做可以降低账户类的复杂性,提高计算数值按日累加之和的代码的可复用性,也就是说日后如果需要计算某个其他数值的按日累加之和,可以直接将这个类拿来用。我们将这个类命名为 Accumulator,该类包括 3 个数据成员——表示被累加数值上次变更日期的 lastDate,被累加数值当前值 value 以及到上次变更被累加数值为止的按日累加总和 sum,该类包括 4 个成员函数——一构造函数,用来计算到指定日期的累加结果的函

数 getSum,用来在指定日期更改数值的 change 以及用来将累加器清零并重新设定初始日期和数值的 reset。

//代码内容 1：指针和字符串的使用

```
//date.h
#ifndef __DATE_H__
#define __DATE_H__
class Date {   //日期类
private:
    int year;       //年
    int month;          //月
    int day;        //日
    int totalDays;      //该日期是从公元元年 1 月 1 日开始的第几天
public:
    Date(int year, int month, int day);   //用年、月、日构造日期
    int getYear() const { return year; }
    int getMonth() const { return month; }
    int getDay() const { return day; }
    int getMaxDay() const;              //获得当月有多少天
    bool isLeapYear() const {   //判断当年是否为闰年
        return year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
    }
    void show() const;                  //输出当前日期
    //计算两个日期之间差多少天
    int distance(const Date& date) const {
        return totalDays - date.totalDays;
    }
};
#endif //__DATE_H__

//date.cpp
#include "date.h"
#include <iostream>
#include <cstdlib>
using namespace std;
namespace { //namespace 使下面的定义只在当前文件中有效
    //存储平年中的某个月 1 日之前有多少天，为便于 getMaxDay 函数的实现，该数组多出一项
    const int DAYS_BEFORE_MONTH[] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
}
Date::Date(int year, int month, int day) : year(year), month(month), day(day) {
```

```
        if (day <= 0 || day > getMaxDay()) {
            cout << "Invalid date: ";
            show();
            cout << endl;
            exit(1);
        }
        int years = year - 1;
        totalDays = years * 365 + years / 4 - years / 100 + years / 400
                    + DAYS_BEFORE_MONTH[month - 1] + day;
        if (isLeapYear() && month > 2) totalDays++;
    }
    int Date::getMaxDay() const {
        if (isLeapYear() && month == 2)
            return 29;
        else
            return DAYS_BEFORE_MONTH[month]-
DAYS_BEFORE_MONTH[month - 1];
    }
    void Date::show() const {
        cout << getYear() << "-" << getMonth() << "-" << getDay();
}

    //accumulator.h
    #ifndef __ACCUMULATOR_H__
    #define __ACCUMULATOR_H__
    #include "date.h"
    class Accumulator {    //将某个数值按日累加
    private:
        Date lastDate;     //上次变更数值的时期
        double value;      //数值的当前值
        double sum;        //数值按日累加之和
    public:
        //构造函数，date 为开始累加的日期，value 为初始值
        Accumulator(const Date &date, double value)
                : lastDate(date), value(value), sum(0) { }
        //获得到日期 date 的累加结果
        double getSum(const Date &date) const {
            return sum + value * date.distance(lastDate);
        }
        //在 date 将数值变更为 value
        void change(const Date &date, double value) {
            sum = getSum(date);
            lastDate = date; this->value = value;
        }
```

```
        //初始化，将日期变为 date，数值变为 value，累加器清零
        void reset(const Date &date, double value) {
            lastDate = date; this->value = value; sum = 0;
        }
    };
#endif //__ACCUMULATOR_H__

    //account.h
    #ifndef __ACCOUNT_H__
    #define __ACCOUNT_H__
    #include "date.h"
    #include "accumulator.h"
    #include <string>
    class Account { //账户类
    private:
        std::string id;//帐号
        double balance;  //余额
        static double total; //所有账户的总金额
    protected:
        //供派生类调用的构造函数，id 为账户
        Account(const Date &date, const std::string &id);
        //记录一笔帐，date 为日期，amount 为金额，desc 为说明
        void record(const Date &date, double amount, const std::string &desc);
        //报告错误信息
        void error(const std::string &msg) const;
    public:
        const std::string &getId() const { return id; }
        double getBalance() const { return balance; }
        static double getTotal() { return total; }
        //显示账户信息
        void show() const;
    };
    class SavingsAccount : public Account { //储蓄账户类
    private:
        Accumulator acc;//辅助计算利息的累加器
        double rate;        //存款的年利率
    public:
        //构造函数
        SavingsAccount(const Date &date, const std::string &id, double rate);
        double getRate() const { return rate; }
        //存入现金
        void deposit(const Date &date, double amount, const std::string &desc);
        //取出现金
        void withdraw(const Date &date, double amount, const std::string &desc);
```

```
    void settle(const Date &date); //结算利息，每年 1 月 1 日调用一次该函数
};
class CreditAccount : public Account { //信用账户类
private:
    Accumulator acc;//辅助计算利息的累加器
    double credit;          //信用额度
    double rate;        //欠款的日利率
    double fee;             //信用卡年费
    double getDebt() const {   //获得欠款额
        double balance = getBalance();
        return (balance < 0 ? balance : 0);
    }
public:
    //构造函数
    CreditAccount(const Date &date, const std::string &id, double credit,
double rate, double fee);
    double getCredit() const { return credit; }
    double getRate() const { return rate; }
    double getFee() const { return fee; }
    double getAvailableCredit() const { //获得可用信用
        if (getBalance() < 0)
            return credit + getBalance();
        else
            return credit;
    }
    //存入现金
    void deposit(const Date &date, double amount, const std::string &desc);
    //取出现金
    void withdraw(const Date &date, double amount, const std::string &desc);
    void settle(const Date &date); //结算利息和年费，每月 1 日调用一次该函
数
    void show() const;
};
#endif //__ACCOUNT_H__


//account.cpp
#include "account.h"
#include <cmath>
#include <iostream>
using namespace std;
double Account::total = 0;
//Account 类的实现
Account::Account(const Date &date, const string &id)
```

```
        : id(id), balance(0) {
    date.show(); cout << "\t#" << id << " created" << endl;
}
void Account::record(const Date &date, double amount, const string &desc) {
    amount = floor(amount * 100 + 0.5) / 100;  //保留小数点后两位
    balance += amount; total += amount;
    date.show();
    cout << "\t#" << id << "\t" << amount << "\t" << balance << "\t" << desc
<< endl;
}
void Account::show() const { cout << id << "\tBalance: " << balance; }
void Account::error(const string &msg) const {
    cout << "Error(#" << id << "): " << msg << endl;
}
//SavingsAccount 类相关成员函数的实现
SavingsAccount::SavingsAccount(const Date &date, const string &id, double
rate)
        : Account(date, id), rate(rate), acc(date, 0) { }
void SavingsAccount::deposit(const Date &date, double amount, const string
&desc) {
    record(date, amount, desc);
    acc.change(date, getBalance());
}
void SavingsAccount::withdraw(const Date &date, double amount, const string
&desc) {
    if (amount > getBalance()) {
        error("not enough money");
    } else {
        record(date, -amount, desc);
        acc.change(date, getBalance());
    }
}
void SavingsAccount::settle(const Date &date) {
    double interest = acc.getSum(date) * rate    //计算年息
                        / date.distance(Date(date.getYear() - 1, 1, 1));
    if (interest != 0) record(date, interest, "interest");
    acc.reset(date, getBalance());
}
//CreditAccount 类相关成员函数的实现
CreditAccount::CreditAccount(const Date& date, const string& id, double credit,
double rate, double fee)
        : Account(date, id), credit(credit), rate(rate), fee(fee), acc(date, 0) { }
void CreditAccount::deposit(const Date &date, double amount, const string
&desc) {
```

```
        record(date, amount, desc);
        acc.change(date, getDebt());
    }
    void CreditAccount::withdraw(const Date &date, double amount, const string &desc) {
        if (amount - getBalance() > credit) {
            error("not enough credit");
        } else {
            record(date, -amount, desc);
            acc.change(date, getDebt());
        }
    }
    void CreditAccount::settle(const Date &date) {
        double interest = acc.getSum(date) * rate;
        if (interest != 0) record(date, interest, "interest");
        if (date.getMonth() == 1)
            record(date, -fee, "annual fee");
        acc.reset(date, getDebt());
    }
    void CreditAccount::show() const {
        Account::show();
        cout << "\tAvailable credit:" << getAvailableCredit();
}

    //main.cpp
    #include "account.h"
    #include <iostream>
    using namespace std;
    int main() {
        Date date(2008, 11, 1);     //起始日期
        //建立几个账户
        SavingsAccount sa1(date, "S3755217", 0.015);
        SavingsAccount sa2(date, "02342342", 0.015);
        CreditAccount ca(date, "C5392394", 10000, 0.0005, 50);
        //11 月份的几笔账目
        sa1.deposit(Date(2008, 11, 5), 5000, "salary");
        ca.withdraw(Date(2008, 11, 15), 2000, "buy a cell");
        sa2.deposit(Date(2008, 11, 25), 10000, "sell stock 0323");
        //结算信用卡
        ca.settle(Date(2008, 12, 1));
        //12 月份的几笔账目
        ca.deposit(Date(2008, 12, 1), 2016, "repay the credit");
        sa1.deposit(Date(2008, 12, 5), 5500, "salary");
        //结算所有账户
```

```
    sa1.settle(Date(2009, 1, 1));
    sa2.settle(Date(2009, 1, 1));
    ca.settle(Date(2009, 1, 1));
    //输出各个账户信息
    cout << endl;
    sa1.show(); cout << endl;
    sa2.show(); cout << endl;
    ca.show(); cout << endl;
    cout << "Total: " << Account::getTotal() << endl;
    return 0;
}
```

三、 实验要求

1、 完成 PTA 作业《C++语言程序设计工程实践：面向过程高级》的所有实验题。

2、 成功在 Visual Studio 2019 里面运行上述代码。